Real-Time GPU Implementation of Transverse Oscillation Vector Velocity Flow Imaging

Bradway, David; Pihl, Michael Johannes; Krebs, Andreas; Tomov, Borislav Gueorguiev; Kjær, Carsten Stråsø; Nikolov, Svetoslav Ivanov; Jensen, Jørgen Arendt

Published in:
Proceedings of the SPIE - Progress in Biomedical Optics and Imaging

Link to article, DOI:
10.1117/12.2043582

Publication date:
2014

Document Version
Early version, also known as pre-print

Paper presented at the SPIE Medical Imaging 2014:

Real-Time GPU Implementation of Transverse Oscillation Vector Velocity Flow Imaging

David Pierson Bradway, Michael Johannes Pihl, Andreas Krebs, Borislav Gueorguiev Tomov, Carsten Stråsø Kjær, Svetoslav Ivanov Nikolov and Jørgen Arendt Jensen

Center for Fast Ultrasound Imaging,
Biomedical Engineering Group,
Department of Electrical Engineering
Ørsteds Plads Building 349,
Technical University of Denmark,
2800 Kgs. Lyngby, Denmark.

To be published in the 2014 SPIE Medical Imaging Proceedings.

Real-time GPU implementation of transverse oscillation vector velocity flow imaging

David Pierson Bradway[a], Michael Johannes Pihl[a], Andreas Krebs[a], Borislav Gueorguiev Tomov[a], Carsten Stråsø Kjær[b], Svetoslav Ivanov Nikolov[b], and Jørgen Arendt Jensen[a]

[a]Center for Fast Ultrasound Imaging, Department of Electrical Engineering, Technical University of Denmark, DK-2800 Lyngby, Denmark;
[b]BK Medical ApS, Herlev, Denmark

ABSTRACT

Rapid estimation of blood velocity and visualization of complex flow patterns are important for clinical use of diagnostic ultrasound. This paper presents real-time processing for two-dimensional (2-D) vector flow imaging which utilizes an off-the-shelf graphics processing unit (GPU). In this work, Open Computing Language (OpenCL) is used to estimate 2-D vector velocity flow *in vivo* in the carotid artery. Data are streamed live from a BK Medical 2202 Pro Focus UltraView Scanner to a workstation running a research interface software platform. Processing data from a 50 millisecond frame of a duplex vector flow acquisition takes 2.3 milliseconds seconds on an Advanced Micro Devices Radeon HD 7850 GPU card. The detected velocities are accurate to within the precision limit of the output format of the display routine. Because this tool was developed as a module external to the scanner's built-in processing, it enables new opportunities for prototyping novel algorithms, optimizing processing parameters, and accelerating the path from development lab to clinic.

Keywords: diagnostic ultrasound, velocity estimation, vector velocity flow imaging, VFI transverse oscillation method, GPU, real-time processing OpenCL

1. INTRODUCTION

Compared to magnetic resonance (MR) imaging and computed tomography (CT), the relative strengths of clinical diagnostic ultrasound include real-time guidance of procedures and rapid, inexpensive examinations. In a research and development setting, live feedback is important for guiding experiments, ensuring alignment, and visualizing temporal effects as they occur. When designing and evaluating new imaging techniques or processing schemes, the iteration process can be accelerated by avoiding "flying blind" and relying on off-line analysis. Real-time visualization and analysis can enable rapid prototyping of novel techniques and can help shorten the time to clinical translation.

Real-time estimation of blood velocity is an important clinical tool, and color flow mapping (CFM) based on the autocorrelation approach[1] is a commonly used. In the conventional form, CFM mode displays 1-D axial velocities acquired and estimated in a 2-D region of interest. But this single axial component of velocity along the ultrasound beam direction does not provide a complete representation of the true 2-D velocities in the scanning plane.

Cost-savings and miniaturization trends in the diagnostic ultrasound industry have increasingly shifted more operations from hardware like application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), and digital signal processors (DSPs) toward software processing in PC platform components like central processing units (CPUs) and graphics processing units (GPUs). Compared to B-Mode image processing, CFM has higher computational complexity, requiring averaging, autocorrelation, discrimination between flow and tissue. This complexity led to research on implementing CFM processing on a GPU for standard 1-D axial velocities in a 2-D region of interest.[2]

Further author information: Send correspondence to D. P. Bradway. E-mail: bradway@elektro.dtu.dk

With the introduction of vector velocity techniques, like the Transverse Oscillation (TO) method, as outlined by Jensen and Munk,[3,4] estimates of both transverse (v_x) and axial (v_z) velocity components can be made simultaneously. A dual-peaked receive apodization is used for the TO method, creating a double-oscillating field sensitive to full vector motion in the imaging plane. Some commercial systems (such as 2202 Pro Focus UltraView Scanner, BK Medical, Herlev, Denmark) are now capable of estimating in-plane velocity direction and magnitude, and some recent studies have utilized this commercial implementation called Vector Flow Imaging (VFI).[5,6] The VFI package relies on a fixed analysis routine, which hinders further optimization or fine tuning of the algorithm and parameters.

Other studies using the TO method have relied on off-line analysis of data acquired with the synthetic aperture real-time ultrasound scanner (SARUS).[7] These off-line analysis studies allow for custom processing and optimized parameters, but they rely on post-processing and thus do not provide real-time vector flow results for guidance and feedback.

To address these limitations on acquiring vector flow data, a custom C++ and Open Compute Language (OpenCL) module was developed for processing 2-D flow data. OpenCL is a flexible programming framework maintained by the Khronos Group (Beaverton, OR) for computation using a many types of hardware including graphical processing units (GPUs). This processing module was fit into a real-time framework provided by BK Medical[8] which allows for custom plug-in processing modules configured with an extensible markup language (XML) file.

In this paper, Section 2 describes the methods and materials used for the development, testing and demonstration of the real-time module. In Section 3, the results are presented and discussed. Finally, Section 4 presents the conclusions and the impact of this work.

2. MATERIALS AND METHODS

The following section describes the methods and materials used in this work for creating and testing the processing module, for acquiring the ultrasound data, and for studying the timing and accuracy of the processing module and results.

2.1 Processing module

The C++ interface to the processing module was developed using Visual Studio 2010 (Microsoft, Redmond, WA). AMD CodeXL (Advanced Micro Devices, Inc., Sunnyvale, CA) was used for debugging and profiling the OpenCL kernels. The module was tested on a desktop computer built from consumer-grade components, including a quad-core Intel i7-3770 CPU, 8 GB of DDR3 SDRAM, and an AMD Radeon HD 7850 graphics card with 2GB GDDR5 and 1024 stream processors.

The first OpenCL kernel unwrapped the input data stream into the axial and transverse lines and reshaped the data into the following order: axial depth samples, lateral positions and sequential shots in the emission sequence. Following the data unwrapping, mean stationary echo-canceling (clutter filtering) was performed by subtracting the mean ensemble value of the N_l lines. For for axial and lateral dimensions, the autocorrelation was found across the emissions as outlined in previous work.[4] Next the arctangent was used to find average phase shift, and the results were converted from radians to velocities. Kernels were then used to calculate the maximum and minimum estimated velocities and the standard deviation of the echo signal across emissions, for use in flow discrimination. Finally, data were reshaped and converted into the expected format.

It should be noted that some of the kernels are independent, such as those processing only axial or lateral directions. These kernels could be executed independently, rather than run sequentially. This may enable a speed improvement if the GPU is not fully occupied in any of the kernels.

2.2 Real-time acquisition

A BK Medical Pro Focus scanner and BK8670 linear array were used to acquire frames of B-Mode and VFI in a continuous loop. The right carotid artery of a healthy 27 year-old male volunteer was scanned with a 20 Hz frame rate. The beamformed, demodulated in-phase and quadrature (IQ) samples passed over a camera link to a frame grabber card in a workstation equipped with the UA2227 research interface[8] (BK Medical, Herlev, Denmark).

Table 1. 2-D TO measurement timing results

Process	Duration (ms)	%
Write data buffer to GPU	0.44	19
Unwrap and format input buffer	1.44	62
Find standard deviation	0.03	1.4
Autocorrelate axial data	0.03	1.2
Find axial arctangent	0.01	0.4
Autocorrelate transverse data	0.04	1.6
Find transverse arctangent	0.01	0.5
Find range	0.09	4.0
Wrap and format output buffers	0.01	0.4
Read Z buffer from GPU	0.11	4.6
Read X buffer from GPU	0.11	4.6
Total	2.30	100

The flow dataset contained 28 lateral lines, $N_l = 16$ emissions per estimate, and 208 axial IQ sample pairs per line. For each transmission in the flow sequence, one axial line, one left line, and one right line were beamformed by the Pro Focus scanner and passed to the workstation. The B-Mode data were processed with standard envelope detection and log compression modules, while the flow data were routed to the custom OpenCL module. In addition to the live display of B-Mode and flow estimates, input and output buffers were saved to disk for further off-line analysis and archival.

2.3 Off-line timing and accuracy study

Routines were also tested for speed of processing a dataset that was saved to disk. Thirteen frames of beamformed IQ data were saved and processed off-line using the AMD CodeXL profiler tool. The speed of memory transfers and the timing of each OpenCL kernel routine were tabulated for the 13 frames. The average durations were calculated for the raw data transfers from the CPU host memory to GPU device memory, for result transfers back to the RAM, and for average time spent in each OpenCL kernel routine.

The first of these frames was also examined in terms of the precision of the OpenCL results. The output display from the OpenCL implementation is presently limited in the ultrasound research framework to an 8-bit integer buffer for the X and Z velocity components. To verify the results of the kernels and to test the impact of this limited precision, a Matlab (Mathworks, Natick, MA) implementation of the processing routines was developed which utilized 32-bit, double-precision floating point variables for all steps following the initial import of the 16-bit integer raw IQ data samples.

3. RESULTS AND DISCUSSION

This Section presents results from real-time scanning and from checking the timing and accuracy of the methods when run off-line.

3.1 Real-time acquisition

VFI data were acquired in an *in vivo* carotid artery. Figure 1 illustrates both the B-Mode and 2-D vector flow within a narrower region of interest in the middle of the field of view. The lengths of the arrows indicate the relative velocity magnitude of the flow at those locations, and the colors indicate both the magnitude and direction of flow. Velocity estimates below a user-tuned threshold are masked in the composite image. The color wheel shows the direction and magnitude of the vector velocity estimates. The live image window updates at a rate of 20 Hz as data are streamed from the Pro Focus scanner.

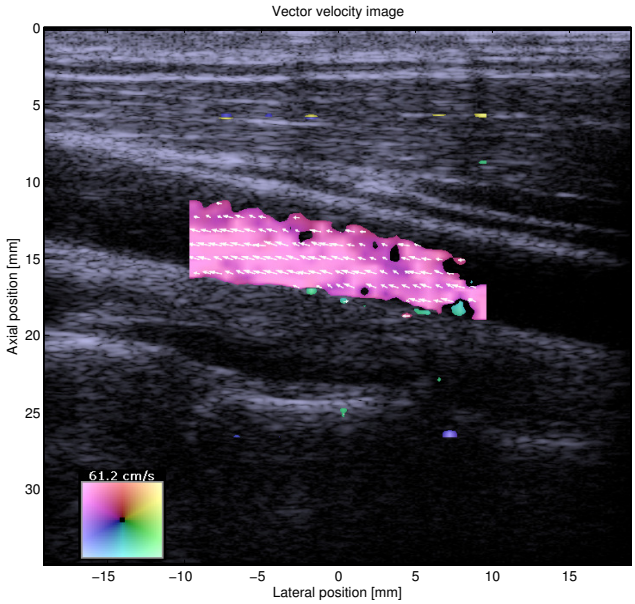

Figure 1. An example image showing B-Mode with 2-D velocity vectors at the center of a carotid artery in a healthy 27 year old volunteer. The color coding corresponds a full 360° color map, and the lengths of the arrows indicate the relative magnitudes of the flow.

3.2 Off-line timing and accuracy study

Table 1 shows processing times averaged over 13 frames of flow data, where each was acquired in 50 milliseconds. For the first frame, the time to write data to the GPU was 5.32 milliseconds. This time was omitted in the averaging because it was observed to be a transient effect of starting the processing routine and did not accurately represent the steady-state operation as would be encountered in live scanning and stream processing. For the rest of the frames, the average time duration for data transfer and processing was 2.3 milliseconds.

The OpenCL kernel operation that takes the most time is the unwrapping and reshaping of the input data buffers. Though the entire process already meets the demands for real-time processing, this unwrapping operation could be eliminated with appropriate indexing into the data buffers, at the expense of programming convenience and readability.

The time constraints for real-processing are set by the duration of the acquisition frame which is determined by the total number of transmits, the pulse repetition frequency, and B-Mode acquisition time. The absolute time required for processing a frame of data scales with dimensions, such as axial sampling rate, emissions per estimate, and lateral locations. In the example sequence used in this work, the data were processed over 20 times faster than acquired. Extra data processing algorithms could be incorporated, or more data could be acquired and processed and still meet the requirements for real-time display.

The first frame of the sequence was examined to compare the accuracy of the OpenCL routine to a 32-bit Matlab implementation of the same methods. Figure 2 shows an example segment from the lateral velocity results for OpenCL and Matlab. The y-axis has been scaled to the quantization limit of the signed 8-bit integers of the OpenCL results. The Matlab results are not restricted to the 8-bit scale and have full floating point values. The trend in this segment suggests the rounding to 8-bit is accomplished with low bias, as the OpenCL result seems to be randomly distributed about the Matlab value. In the final OpenCL kernel, when the data are converted to 8-bit integer format, the estimates are rounded to the nearest bit. The values are uniformly distributed across the range that is truncated.

The point-wise differences across the whole frame of results were taken between the 8-bit OpenCL and 32-bit Matlab estimes. The mean difference, or bias, in the X and Z velocities are 1.1% and -2.5% of the quantization step size which is 1/256 of the entire velocity range. This bias is extremely low, but this result highlights the

importance of filling the dynamic range of with useful data by considering the speed of expected flow in the region of interest and by setting the aliasing limit and pulse repetition frequency appropriately,

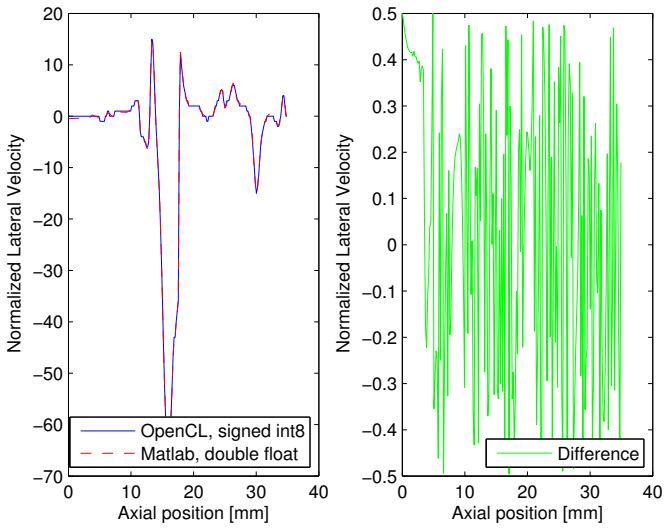

Figure 2. A result segment through depth where the center of the vessel appears around 15 millimeters. These lateral velocity estimates have been normalized by the 8-bit quantization level of the OpenCL routine, the results of which are shown in the left plot in blue. The red dashed line is the higher-precision Matlab velocity estimates, which lies nearly perfectly on top of the OpenCL estimates. The green line in the right plot is the difference between the two curves, and it shows error of ± 0.5 the quantization step size.

Over the full range of displayed velocities, the 8-bit quantization level does seem sufficient for representing arrow vector sizes and color coding intensity levels, but it will limit the precision of data saved to disk or used in subsequent algorithms in the signal processing chain, such as calculating pressure from the velocity estimates via the Navier-Stokes equations.[9] This limitation should be addressed by future updates to processing module in the research interface. An intermediate buffer will be added in the processing chain to preserve the higher bit depth of the velocity estimates for use by subsequent routines or saving to disk. A rounding module will be added to take the high bit depth velocities and provide the 8-bit estimates that are expected by the display routine.

This 2-D method is an improvement over conventional axial estimation, but it does not provide the full 3-D velocity vector, including the out-of-plane elevation component. For that purpose, the TO method was extended to 3-D, and feasibility was demonstrated using a 2-D matrix array transducer in both flow phantoms and *in vivo* measurements.[7] So far, processing for 3-D Vector Flow Imaging has been completed off-line in Matlab due to a lack of both real-time processing and commercial scanners which support using 2-D ultrasound matrix arrays with VFI. The real-time processing method introduced in this work will soon be adapted for use with live 3-D flow data from the SARUS scanner.

4. CONCLUSION AND PERSPECTIVES

A real-time implementation of 2-D vector velocity flow imaging has been demonstrated which utilizes GPU processing. Timing results from an example data set showed that the processing is quite fast, at 20 times the acquisition rate, and the results are accurate to within the present quantization limit of the 8-bit integer output.

The use of an off-the-shelf graphics card enables straightforward inclusion into the processing chain of an experimental laboratory setup or a commercial ultrasound scanner and reduces the reliance on expensive custom circuit boards. These real-time tools enable estimation of quantitative volumetric flow rates and bed-side examinations with angle-independent flow estimation. This functionality is important for diagnosing cardiovascular disease, visualizing complex flow patterns, and improving understanding of cardiovascular pathology.

ACKNOWLEDGMENTS

This work was supported by BK Medical ApS, Denmark, by grants 024-2008-3 and 82-2012-4 from the Danish Advanced Technology Foundation, and by a postdoctoral scholarship from the Whitaker International Program, administered by the Institute of International Education (IIE).

REFERENCES

[1] Kasai, C., Namekawa, K., Koyano, A., and Omoto, R., "Real-Time Two-Dimensional Blood Flow Imaging using an Autocorrelation Technique," *IEEE Trans. Son. Ultrason.* **32**, 458–463 (1985).

[2] Chang, L. W., Hsu, K. H., and Li, P. C., "Graphics processing unit-based high-frame-rate color doppler ultrasound processing," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **56**, 1856–1860 (September 2009).

[3] Jensen, J. A. and Munk, P., "A New Method for Estimation of Velocity Vectors," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **45**, 837–851 (1998).

[4] Jensen, J. A., "A New Estimator for Vector Velocity Estimation," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **48**(4), 886–894 (2001).

[5] Hansen, K. L., Pedersen, M. M., Møller-Sørensen, H., Kjaergaard, J., Nilsson, J. C., Lund, J. T., Jensen, J. A., and Nielsen, M. B., "Intraoperative cardiac ultrasound examination using vector flow imaging," *Ultrason. Imaging* **35**, 318–332 (Oct 2013).

[6] Pedersen, M. M., Pihl, M. J., Haugaard, P., Hansen, J. M., Hansen, K. L., Nielsen, M. B., and Jensen, J. A., "Comparison of real-time in vivo spectral and vector velocity estimation," *Ultrasound Med. Biol.* **38**(1), 145–151 (2012).

[7] Pihl, M. J. and Jensen, J. A., "Measuring 3D velocity vectors using the transverse oscillation method," in [*Proc. IEEE Ultrason. Symp.*], 1881–1885 (2012).

[8] Hemmsen, M. C., Nikolov, S. I., Pedersen, M. M., Pihl, M. J., Enevoldsen, M. S., Hansen, J. M., and Jensen, J. A., "Implementation of a versatile research data acquisition system using a commercially available medical ultrasound scanner," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.* **59**(7), 1487–1499 (2012).

[9] Olesen, J. B., Traberg, M. S., Pihl, M. J., and Jensen, J. A., "Non-invasive measurement of pressure gradients using ultrasound," in [*Proc. SPIE Med. Imag.*], 1–7 (mar 2013). 86750G.